Middle East Technical University Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring 2019-2020

Prolog Mini Homework

Start date: 03 06 2020, Wednesday, 09:00
Due date: 04 06 2020, Wednesday, 09:00

1 Objective

This homework aims to familiarize you with logic programming concepts by implementing some predicates on a given knowledge base using Prolog.

Keywords: *logic programming, Prolog*

2 Problem Definition

In this homework, you are going to implement predicates that operate on a knowledge base derived from a popular AutoChess game called Dota Underlords. Autochess is a round based battle simulation game genre. In each round, player will buy units from a limited selection shop and place them on a chess board. After this placement, the units will fight with the opponent's units to determine the winner. In this homework, we will only be focusing on units and some of their properties. The properties we will be interested in, is given below:

- **Name:** This is the unique name of the unit
- **Cost:** Each unit has a corresponding cost to buy
- **Attributes:** Each unit has specific attributes.
- **Star:** This property represents the level of the unit. This will only apply when the unit is active. Units can have at most three stars. Every unit that is bought, starts with a single star. Three units with same star count upgrades to the next star level. This means that, three 1 star units will combine and form a single 2 star unit. Similarly, three 2 star units will combine into a single 3 star unit.

2.1 Knowledge Base Predicate

The `unit` predicate is used to define units and their properties. Its format is given below:

```
unit(Name, Cost, [Attributes]).
```

The attributes are given in no particular order. All units are present in the knowledge base Prolog file provided for you. Examples can be seen below:

```
unit("Tusk", 1, [savage, warrior]).
unit("Storm Spirit", 2, [spirit, mage]).
unit("Windranger", 2, [hunter, vigilant]).
unit("Sven", 4, [human, scaled, knight]).
unit("Lich", 5, [heartless, mage]).
unit("Warlock", 1, [bloodbound, warlock, healer]).
unit("Axe", 5, [brawny, brute]).
```

3 Specifications

In this mini homework, you are going to implement three predicates with varying difficulty.

3.1 The cost Predicate - 30 pts

The cost predicate has two arguments. The first argument is a list of active units and the second argument should be the total cost of units in that list. Active unit is a structure in the given format:

```
active_unit(Name, Star)
```

You are only expected to calculate and check the cost of the units on that list. The generation of a list from cost is **NOT** expected from you. The cost of a single active unit can be calculated using the formula:

$$Cost(Active\_Unit) = 3^{Star-1} \times Cost(Unit)$$

Format of the predicate given below:

```
cost([Active_Unit], Cost).
```

Examples:

```
?- cost([], Cost).
Cost = 0
?- cost([active_unit("Warlock", 1)], Cost).
Cost = 1.
?- cost([active_unit("Warlock", 2)], Cost).
Cost = 3.
?- cost([active_unit("Warlock", 3)], Cost).
Cost = 9.
?- cost([active_unit("Axe", 1)], Cost).
Cost = 5.
?- cost([active_unit("Axe", 2)], Cost).
Cost = 15.
?- cost([active_unit("Axe", 3)], Cost).
Cost = 45.
?- cost([active_unit("Warlock", 2), active_unit("Axe", 3)], Cost).
Cost = 48.
?- cost([active_unit("Warlock", 2), active_unit("Axe", 3), active_unit("Slardar", 1),
active_unit("Troll Warlord", 2), active_unit("Drow Ranger", 2)], Cost).
Cost = 68.
```

3.2 The sort_units Predicate - 35 pts

The sort_units predicate has two arguments. First argument is a list of unit names and the second argument should be the sorted list based on the cost of the units in a descending order. Sorting should be stable and there is no need to find all possible sorted versions of the first list. Only a single list is enough. Stable sorting means that if the units have the same cost, do not switch their positions. (Hint: Use accumulation(stack))
Format of the predicate given below:

```
sort_units([Unit_Names], (Sorted)[Unit_Names]).
```

Examples:

```
?- sort_units([], L).
Result = [] .
?- sort_units(["Warlock", "Drow Ranger"], Result).
Result = ["Warlock", "Drow Ranger"] .
?- sort_units(["Tiny", "Enigma"], Result).
Result = ["Enigma", "Tiny"] .
?- sort_units(["Batrider", "Treant Protector", "Dazzle"], Result).
Result = ["Treant Protector", "Dazzle", "Batrider"] .
?- sort_units(["Batrider", "Treant Protector", "Mirana", "Dragon Knight",
"Dazzle"], Result).
Result = ["Dragon Knight", "Mirana", "Treant Protector", "Dazzle", "Batrider"].
?- sort_units(["Warlock", "Drow Ranger", "Axe", "Faceless Void", "Lycan", "Luna",
"Dazzle", "Tusk", "Razor"], Result).
Result = ["Axe", "Faceless Void", "Lycan", "Luna", "Dazzle", "Warlock", "Drow Ranger",
"Tusk", "Razor"] .
?- sort_units(["Warlock", "Axe", "Luna", "Dazzle", "Tusk", "Razor", "Shadow Fiend",
"Bristleback", "Mirana"], Result).
Result = ["Axe", "Mirana", "Shadow Fiend", "Luna", "Dazzle", "Bristleback", "Warlock",
"Tusk", "Razor"] .
```

3.3 The buyable Predicate - 35 pts

The buyable predicate takes four arguments. First argument is a list of unit names available to buy and the second argument is the amount of money you can spend. Third argument should be units than can be bought using the money in the second argument. Fourth predicate is the amount of money left after buying the units. It should be bigger than or equal to 0. Your predicate should be able to find all possible unit list and remaining money combinations when tested .Format can be seen below:

```
buyable([Units], Money, [Bought Units], RemainingMoney).
```

Examples:

```
?- buyable(["Warlock", "Axe"], 50, B, M).
B = ["Warlock"],
M = 49 ;
B = ["Warlock", "Axe"],
M = 44 ;
B = ["Axe"],
M = 45 ;
false.
?- buyable(["Mirana", "Axe", "Lycan", "Broodmother", "Nyx", "Io", "Lifestealer",
"Doom"], 5, B, M).
B = ["Mirana"],
M = 1 ;
B = ["Mirana", "Nyx"],
M = 0 ;
B = ["Axe"],
M = 0 ;
B = ["Lycan"],
M = 2 ;
B = ["Lycan", "Nyx"],
M = 1 ;
B = ["Broodmother"],
M = 2 ;
B = ["Broodmother", "Nyx"],
M = 1 ;
B = ["Nyx"],
M = 4 ;
B = ["Nyx", "Io"],
M = 1 ;
B = ["Nyx", "Lifestealer"],
M = 1 ;
B = ["Nyx", "Doom"],
M = 0 ;
B = ["Io"],
M = 2 ;
B = ["Lifestealer"],
M = 2 ;
B = ["Doom"],
M = 1 ;
false.
```

```
?- buyable(["Mirana", "Axe", "Lycan", "Broodmother"], 10, B, M).
B = ["Mirana"],
M = 6 ;
B = ["Mirana", "Axe"],
M = 1 ;
B = ["Mirana", "Lycan"],
M = 3 ;
B = ["Mirana", "Lycan", "Broodmother"],
M = 0 ;
B = ["Mirana", "Broodmother"],
M = 3 ;
B = ["Axe"],
M = 5 ;
B = ["Axe", "Lycan"],
M = 2 ;
B = ["Axe", "Broodmother"],
M = 2 ;
B = ["Lycan"],
M = 7 ;
B = ["Lycan", "Broodmother"],
M = 4 ;
B = ["Broodmother"],
M = 7 ;
false.
```

4 Regulations

- **Programming Language:** You must code your program in Prolog. Your submission will be run with `swipl` on Cengclass. You are expected make sure your code runs successfully with `swipl` on CengClass. If your submission has compilation errors (that can be fixed trivially), 5 points will be deducted from your final grade.
- **Modules:** You are not allowed to import any modules. However, you are allowed to use the base predicates present in prolog or define your own predicates.
- **Cheating:** Using code from any source other than your own is considered cheating. This includes but not limited to internet sources, previous homeworks, and friends. In case of cheating, the university regulations will be applied.
- **Grading:** You can evaluate your code interactively on Cengclass. However, note that the evaluation test cases are only provided to assist you and are different from the test cases that will be used during grading. Thus, the grade resulting from the evaluation is not your final grade.
- **Late Submission:** There is no late submission.

5 Submission

Submission will be done via Cengclass. You are expected to either edit the file hw.pl directly on Cengclass or upload it after working on your computer.